Highly Parallel Linpack on the IBM p690 *

Mark R. Fahey
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 4500N
Oak Ridge, TN 37831-6203
(faheymr@ornl.gov)

September 25, 2002

Abstract

Oak Ridge National Laboratory installed 27 32-way IBM pSeries 690 SMP nodes in early 2002. In this paper, we describe the Highly Parallel Linpack results on this machine.

1 Introduction

It is incumbent on DOE to be aware of the performance of new or beta systems from high-performance-computing vendors that will determine the performance of future production-class offerings. It is equally important that DOE work with vendors in finding solutions that will fulfill DOE's computational requirements. Thus, ORNL has acquired 27 pSeries 690 (p690) nodes from IBM each equipped with 32 1.3 GHz POWER4 processors.

One of the tests used to determine the performance of this high-performance computer is the Linpack Benchmark. The Linpack Benchmark is a measure of a computer's floating-point rate of execution. It is determined by solving a dense system of linear equations. This performance does not reflect the overall performance of a given system, as no single number ever can. It does, however, reflect the performance of a dedicated system for solving a dense system of linear equations. It is usually regarded as a good correction of peak performance of a system.

The Parallel Linpack Benchmark can be run using any parallel factorization and solver routines. The Highly Parallel Linpack (HPL) [1] software package is provided for just this. The HPL software package solves a (random) dense linear system of equations in double precision arithmetic on distributed-memory computers. HPL solves a linear system of equations by computing the LU factorization with partial row pivoting of the n by n+1 coefficient matrix [A b]. The data is mapped onto a two-dimensional P by Q grid of processors to ensure load balancing using a block-cyclic distribution [2].

2 System

We describe the specifications of the IBM pSeries 690 systems running IBM AIX 5.1D, where each system (node) has 32 1.3GHz POWER4 processors. Each processor can complete four floating-point operations per cycle resulting in a peak rating of 5.2 GigaFlops. The following information is taken from a paper describing benchmark performance of an early evaluation of the p690 [3].

Each POWER4 chip has two processors. Four chips are connected to form a multi-chip module (MCM), and each system has 4 MCMs (32 processors). The two processors on a chip share a single L2 cache for data

*This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

and instructions, but each processor has its own private L1 data cache and L1 instruction cache. The L1 data caches are 32KB in size and two-way set associative, with a first-in-first-out (FIFO) replacement policy. The FIFO replacement policy can make blocking for the L1 cache unproductive because it limits reuse; data loaded earlier are overwritten even if they were used more recently.

The shared L2 on each chip is 1440KB in size. Sharing the L2 cache may be a disadvantage when running independent processes on each processor, but it may be an advantage when running threads that share an address space. "False sharing", separate processors accessing different data on the same cache line, can improve instead of degrade performance. The L2 cache on each MCM chip has point-to-point access to the L2 caches on the other three.

Each MCM goes through four 128MB L3 caches to memory, where each L3 cache connects to a separate memory controller. Therefore, each L3 cache is associated with a certain region of real memory, not a certain processor. Memory pages are striped by cache line across the four L3 caches and memory controllers, maximizing the cache size and memory bandwidth available to any single processor. In addition, the L3s can cache references to memory on other MCMs. Memory requests form an L3 on one MCM can be met by memory, L3, or L2 on another.

The performance of prefetching is dependent on the configuration of the L3 caches. The Level 3 cache prefetch mode is site customizable at boot time. It can be set to prefetch a full 512B line or not. Here it is set to prefetch a 512B line which improves performance of applications that stream through memory, while degrading the performance of applications with irregular memory-access patterns. The line size for L1 and L2 prefetch is 128B.

All reported observations were performed under AIX 51D with memory affinity. Memory affinity is a performance tuning option available to the p690. A memory pool is constructed for each affinity domain reported by the firmware at system boot time. The system will attempt to satisfy page faults from the memory pool closest to the processor that faulted. When a process tries to touch a small page that does not yet exist, that small page will be allocated on the MCM where the process is running (assuming that there is any free memory on that MCM).

The system under test was configured as 27 32-way nodes; 2 with 128 Gb of memory[1], 5 with 64 Gb of memory, and 20 with 32 Gb of memory. This heterogeneous mix of memory might have affected the performance of the benchmark, but the author does not believe this effect was significant.

All nodes are connected with two planes of IBM SP Switch2. The full system tests make use of this configuration as well as using a single plane, and Gigabit Ethernet. In total, there are 27 nodes connected via a switch network to form a 864 processor machine.

3 Linpack Results

The HPL code contains many possible code paths for its operation. It is left to the user to experimentally determine the optimal set of parameters for a given machine configuration. Thus, many tests were run (not reported here) to experimentally determined the optimal parameters. The results stated here only reflect the experimentally chosen best parameters for a single node (i.e., 32 processors) and for the full system test (i.e., 27 32-way nodes.)

The HPL code was compiled as a 64-bit executable, linked to the threaded ESSL library (esslsmp), and uses either row- or column-major ordering for the process grid[2].

Each test was run through the LoadLeveler batch system during dedicated time. The following environment variables were used for each run.

- MP_SHARED_MEMORY=yes
- MP_EAGER_LIMIT=32k
- MP_BUFFER_MEM=64M
- MP_SINGLE_THREAD=yes

[1] Only 96 Gb of the 128 Gb are addressable at this time.

[2] The default setting is to use column-major ordering for the process grid. This can be changed to row-major ordering by modifying the the routine `HPL_pddriver.f`.

When the HPL code was run, the processes and/or threads were bound in consecutive order to a processor, first within a node and then across nodes[3]. Process/thread binding was utilized because experiments showed that variability in the performance of a p690 node increased after the upgrade to AIX 51D and PTF 8[4] unless processes and threads were bound to processors. Without binding, the author observed variability of up to 10%.

3.1 Single node Linpack tests

For the single node, i.e., 32 processor, Linpack tests, the following parameters were identified to be optimal for the HPL code on the p690.

```
N      :   54000
NB     :     200
PFACT  :   Right
NBMIN  :       8
NDIV   :       2
RFACT  :    Left
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Binary-exchange
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
```

Table 1 shows the results of several single node Linpack tests comparing row- versus column-major grid ordering, various process grid sizes, and MPI-only to MPI plus threads. P is the number of "rows", Q is the number of "columns", and T is the number of threads per task. MPI-only runs are signified by a 1 in the T column.

Table 1: Single node (32 processors) Linpack results.

Grid ordering	P	Q	T	Time	GFlops
Row-major	8	4	1	1128.20	93.1
Row-major	4	8	1	1155.55	90.9
Row-major	4	4	2	1136.20	92.4
Row-major	2	4	4	1172.70	89.5
Row-major	4	2	4	1155.13	90.9
Column-major	8	4	1	1121.57	93.6
Column-major	4	8	1	1156.33	90.8
Column-major	2	4	4	1222.61	85.9
Column-major	4	2	4	1205.30	87.1

The data shows that within a single node the optimal method for running the HPL code is with MPI tasks only. The mixture of MPI and threads results in only a slight degradation in performance. The ordering of the process grid has little affect on the results within a node, but a strong preference for a 8×4 process grid is seen.

Note that tests were also run on 64 Gb and 128 Gb nodes. These tests showed that the 64 Gb nodes performed, on average, approximately 0.5% better than the 32 Gb nodes for the same problem sizes. However, the tests run on the 128 Gb nodes performed approximately 1% worse than the 32 Gb nodes. This is probably due to the fact that only 96 Gb of the node is addressable, and that one of the MCMs on the node therefore does not have local L3 cache.

[3]This was done with a modified `MPI_Init.c` routine provided by Farid Paripa of IBM.

[4]There were several upgrades done simultaneously; AIX 51C to 51D, PTF level 6 to 8, firmware upgrades, and new hardware.

3.2 Full system Linpack tests

For the full system, i.e., 864 processor, Linpack tests, the following parameters were identified to be optimal for the HPL code on the p690.

```
N      :  275000
NB     :     200
P      :       8
Q      :      27
PFACT  :   Right
NBMIN  :       8
NDIV   :       2
RFACT  :    Left
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Binary-exchange
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
```

This input specifies an 8×27 process grid which corresponds to 216 MPI tasks. In addition, the number of threads per task was 4.

Table 2 shows the results of several 864 processor Linpack tests comparing row- and column-major grid ordering, user space and IP protocols, and several switch device settings. These tests were conducted to see the effect of several factors after determining the optimal Linpack rate.

Table 2: Full system (864 processors) Linpack results.

Switch Device	Protocol	Grid Ordering	Time	TFlops
Dual-plane	US	Row-major	6002.82	2.312
Dual-plane	US	Row-major[5]	6197.66	2.237
Single-plane	US	Row-major	6711.69	2.066
Dual-plane	US	Column-major	7892.93	1.757
Dual-plane	IP	Row-major	6191.91	2.239
Gig-E adapters	IP	Row-major	7482.78	1.853

The data shows that dual-plane provides more than a 10% benefit when compared to single-plane which obviously is attributable to increased bandwidth. Although the process grid ordering has little effect in the single node case, for the highly parallel case it is vitally important to have it set properly. For example, a run with a process grid of 32×8 would achieve a little over 1 TFlops, or about half of the optimal rate. It is interesting to note that there is not much performance degradation using IP instead of User Space. Even use of the Gigabit-Ethernet adapters rather than the colony switch results in reasonable performance. Thus, this data suggests at the minimum that bandwidth was not the primary factor limiting the benchmark from obtaining an even greater rating.

3.3 Block size

The key to achieving high performance on modern cache-based processors is to use blocked algorithms that permit reuse of data and that use up all the available memory without paging or without using up memory for the system tasks like MPI. These are important issues on this architecture as performance drops significantly with slight changes in the optimal problem size or block size. The optimal block size is 200, which corresponds

[5]This test used a process grid of 9×24.

to 320 KB. This is nearly 1/4 the size of Level 2 cache. The L2 cache is shared between 2 processors on a p690 and all processors on a node were used for these tests, so the optimal blocksize translates to about 1/2 of the L2 cache per processor.

3.4 Process Grid implications

The experiments showed that the optimal process grid was 8×27 and the optimal ordering of the process grid was row-major. Why is this so? The author expected a 9×24 grid to yield better performance than the 8×27 grid since there is a natural fit of 3 nodes to each process row. The following attempts to answer this question.

To begin, consider Figures 1 and 2. Figure 1 depicts the 8×27 process grid case. In the figure, each color represents a different p690 node with 32 processors. Remember that each MPI task is spawning 4 threads, so only 8 tasks on a node. Process column 10 is highlighted in both Figures 1 and 2 which shows a process column and the nodes it spans. In Figure1, this process column spans 8 nodes (as it does in Figure2), but it spans a different CPU on each of these nodes.

Figure 2 illustrates the 9×24 case, and the natural fit of three nodes per process row is easily seen. Note that process column 10 spans the same CPU[6] of each node unlike in Figure 1. This natural and easy to visualize fit of nodes to process rows and "same numbered CPU" in a process column is important to note because it is the reason for poorer performance on this process grid.

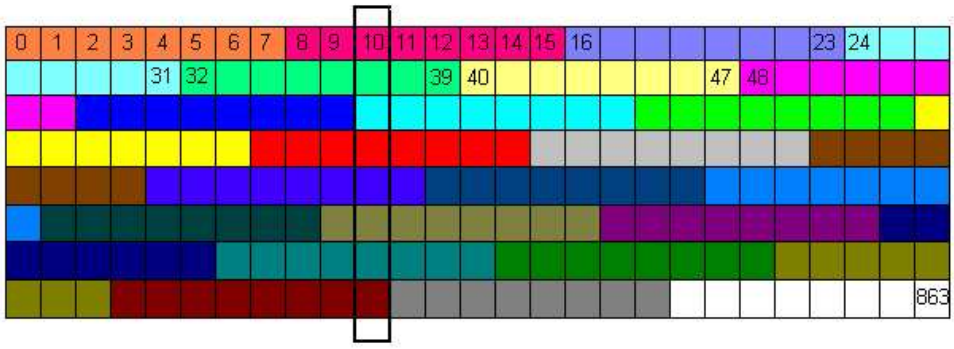

Figure 1: 8×27 process grid.

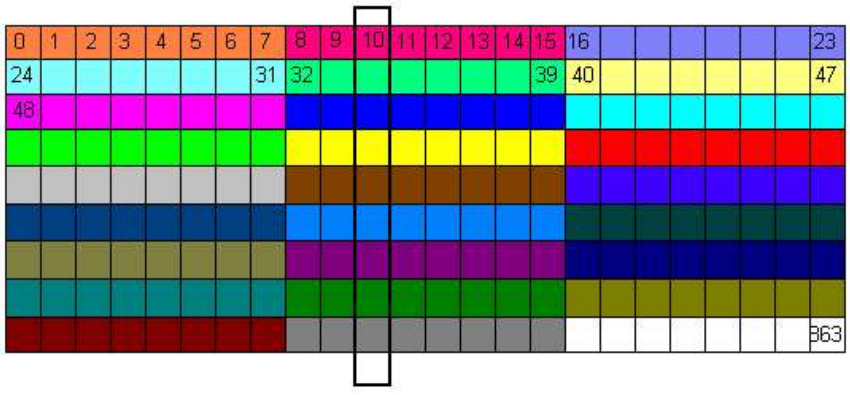

Figure 2: 9×24 process grid.

[6]Since the processes/threads are bound in consecutive order across a node first and then across nodes secondly, it is known that a process column in this cases spans the same CPU over 8 nodes.

Furthermoe, at a given iteration of the main loop of the HPL algorithm, and because of the Cartesian property of the distribution scheme, each panel factorization occurs in one column of processes. The associated swap and broadcast operation of the pivot row are combined into one single communication step for each panel column pivot search. The panel broadcast is a broadcast of the panel factorization to the other process columns. Tests showed that for the panel broadcast, the modified increasing-ring broadcast algorithm performed best. Figure 3 shows the modified increasing-ring algorithm as taken from [1]. Process

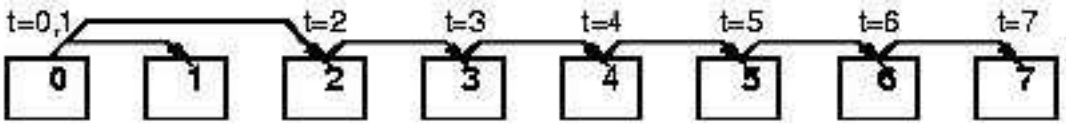

Figure 3: Modified Increasing-Ring algorithm.

0 sends two messages and process 1 only receives one message.

With these facts in mind, arguments are presented to explain the observations.

- A short, wide process grid
  - The process column length (number of process rows) is relatively short, i.e., 8 or 9, and each process is on a separate node. Thus the panel factorization which is computed on a process column only requires the communication between 8 or 9 processes on separate nodes. Thus, a short process grid performs less internode communication than a tall process grid.
  - A row operation is limited to a small number of nodes (4 or 5 for the 8×27 case and 3 for the 9×24 case), so most row communication is fast via shared memory. Thus, a wide process grid on this machine utilizes shared memory communication more than a narrow process grid.
- The 9×24 process grid
  - With the exact fit of 3 nodes to a process row (see Figure 2), the processes should stay relatively synchronized while doing communication across rows or columns. This might lead one to believe that this could reduce wait time for, say the panel broadcasts, because barriers would incur no major penalties. But the processes don't need to be synchronized across rows for the panel broadcast, thus there are no barriers and no benefit due to the naturally ocurring synchronization. The panel broadcasts as performed in separate process rows will either do intra-node communication simultaneously or do inter-node communication simultaneously resulting in an alternating pattern of very light and then very heavy switch traffic. This unbalanced traffic over the switch is probably the reason that the 8×27 process grid is better.
- The 8×27 process grid
  - The panel broadcasts via the MIR algorithm do not make heavy use of the switch at any given time, rather a relatively even load of messages is communicated across the switch. This is because the nodes are staggered compared from one row to the next as they are assigned in the row-major fashion. There is little penalty for the staggered assignment of nodes. Consider column 10 of Figure 1, assuming that, for each row, the process in column 10 is "0", then as it sends data to the right, some processes in column 10 can do it faster than others because some can use shared memory. And this argument can be applied to the remaining sends in MIR. However, on the whole, as the process column is broadcast, it will take approximately the same amount of time across each row. And, the switch is transmitting a very balanced load of data during this broadcast.

In summary, a short fat process grid is better than a tall skinny grid due to the fact that column communication is performed across just a few nodes and that row communication gets a huge benefit from shared memory. Also, the row-major ordering with the 8×27 grid leads to balanced network traffic.

4 Conclusions

The Linpack 1000 Benchmark[7] only achieves approximately 63% of peak on one processor and 56% of peak on a 32-way node. The previous generation of POWER processors (POWER3) could attain roughly 85% of peak on one processor. The difference is that the POWER4 does not have enough rename registers relative to its computational ability as compared to the POWER3. Although the Linpack benchmark is only one number, it shows that the POWER4 relative to its peak performance is not ideal for solving a dense system of equations.

Table 3 shows the percent of peak achieved at 1, 32, and 864 processors. The data shows that there is

Table 3: Percent of Peak

Processors	GFlops	Peak %
1	3.3	63.4
32	93.6	56.2
864	2310	51.4

only a a 12% drop-off from 1 processor to 864 and a 5% drop-off from 32 processors to 864. This indicates the full system run was at best impaired by less than 5% due to bandwidth constraints (i.e., the switch.) The best chance for IBM to improve upon the parallel Linpack rating is to significantly improve the performance of Linpack on one processor; more likely to happen with the next generation of POWER processor.

More importantly though these tests show the importance of using an appropriately chosen processor grid for applications that employ 2-D block-cyclic data decompositions. The best processor grid is going to be application and data dependent, but it will probably be a short, wide grid. Furthermore, an improperly chosen processor grid can mean a degradation of 5 to 50 percent in performance.

5 Acknowledgments

We gratefully acknowledge the aid of the following persons:

- Don Maxwell of ORNL for his efforts in preparing the p690 nodes for the Linpack tests.
- Frank Johnston of IBM for passing on the idea to use row-major ordering first suggested by Antione Petitet.
- Vance Shaffer and Farid Parpia of IBM for supplying the process binding scripts and libraries.

References

[1] Petitet, A., Whaley, R.C., Dongarra, J., *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*, `http://www.netlib.org/benchmark/hpl`, Sep., 2000.

[2] Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley Publishing Company, 1995.

[3] Worley, P., et. al., *Early Evaluation of the IBM p690*, submitted for inclusion in the Proceedings of SC2002.

[7] Linpack 1000 tests were also performed and only the final performance numbers are mentioned here.